Name: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**MAE 493G/ CpE 493M Mobile Robotics, Spring 2017**

**Homework #5 (10 Points)**

**Assigned: 03/22/2017 Due: 03/29/2017**

*Note: please properly document your answers (including MATLAB code) in a   
Microsoft Word file and upload it to* ***eCampus****.*

**Problem #1 (5 Points)**

Consider a linear mass-spring-damper system shown in the figure below:



The underlining dynamics can be modeled with a second order differential equation:



Where *M* = 1kg is the mass, *K* = 5N/m is the spring constant, and *f*v = 1 N-s/m is the damping coefficient. The force on the mass in this case is a sine wave  and the unit is in N. The position of the mass, *x*, is measured with a noisy sensor with zero mean and a standard deviation of 0.2m, the force is measured with a noisy sensor with zero mean and a standard deviation of 0.05N.

1. Program a Kalman filter in MATLAB to estimate the position of the mass.
2. Plot the true position, measured position, and the Kalman Filter estimated position in one figure.

%problem 1

%% Generate Truth and measurements

Mass = 1; %kg

k = 5; %N/m

c = 1; %Ns/m

T\_S = 0.1;

T = 1000; %Run Time in seconds (# of iterations)

Time = zeros(T,1);

AccelTru = zeros(T,1);

VelTru = zeros(T,1);

PosTru = zeros(T,1);

ForceMes = AccelTru;

PosMes = PosTru;

VelEst = VelTru;

PosEst = PosTru;

for t = 1 : T

Time(t) = t\*T\_S;

forceTru(t) = 5\*(sin(Time(t)));

%AccelTru(t) = (forceTru(t)/Mass)-(k\*PosTru(t)/Mass)-(c\*VelTru(t)/Mass);

if t > 1

AccelTru(t) = ((-k\*PosTru(t-1))/Mass)-((c\*VelTru(t-1))/Mass)+((forceTru(t))/Mass);

VelTru(t) = VelTru(t-1) + AccelTru(t) \* T\_S;

PosTru(t) = PosTru(t-1) + VelTru(t) \* T\_S;

end

PosMes(t) = PosTru(t) + normrnd(0,0.2);

ForceMes(t) = (forceTru(t) + normrnd(0,0.05));

end

%% Kalman Filter

X = zeros(T,2)'; %Initialize State Vector

P = eye(2); %Initialize covariance matrix

A = [1,T\_S;-k\*T\_S/Mass,1-c\*T\_S/Mass];

B = [0;T\_S/Mass];

C = [1,0];

Q = [0,0;0,0.05^2]\*(T\_S^2); %Proccess Noise

R = 0.4; %Measurement Noise

for n = 2 : T

%% Prediction Step

X(:,n) = A\*X(:,n-1) + B\*ForceMes(n);

P = A\*P\*A' + Q;

%% Measurement Step

K = P\*C'\*inv(C\*P\*C' + R);

X(:,n) = X(:,n) + K\*(PosMes(n) - C\*X(:,n));

P = (eye(2) - K\*C)\*P;

PosEst(n) = X(1,n);

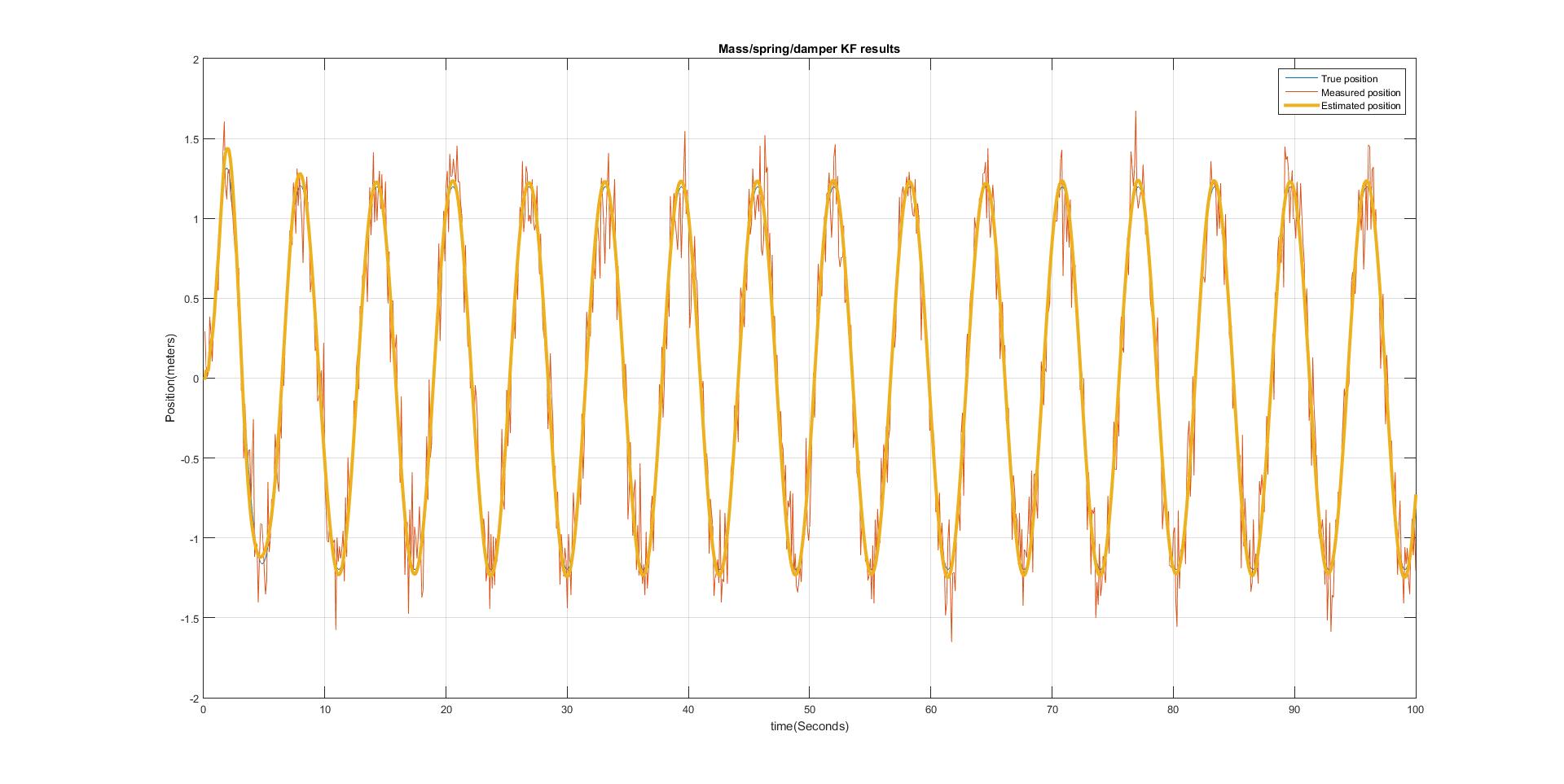
VelEst(n) = X(2,n);

end

figure; plot(Time,PosTru);hold on;plot(Time,PosMes);plot(Time,PosEst,'LineWidth',3);

grid on; legend('True position','Measured position','Estimated position');

xlabel('time(Seconds)'); ylabel('Position(meters)');title('Mass/spring/damper KF results');



**Problem #2 (5 Points)**

Reproduce the plots in the class slides for the simple pendulum (w/ friction) problem. Use the Extended Kalman Filter (EKF) to estimate the pendulum angle. Make sure to write down all the steps including the ones listed already in the class slides.

**Answer:** No sure what is going wrong in the EKF. All the equations seem right, but they are obviously not. I believe the issue is in the prediction step.

%%Problem 2

%% Create truth and mesaurements

m=1;%kg

l=0.5;%m

c=0.5;%coefficient of friction

g=9.81;%m/sec^2

T\_s=0.1;%10 Hz sampling

T=1000;%number of samples

ThetaDDT = zeros(T,1);

ThetaDT = zeros(T,1);

ThetaT = zeros(T,1);

ThetaT(1)=1.6;

HorzDistMes = ThetaDT;

ThetaDE = ThetaDT;

ThetaE = ThetaDT;

ThetaDDT(1)=((-g/l)\*sin(ThetaT(1)))-((c/(m\*l))\*ThetaDT(1));

for t=2:T

ThetaDDT(t)=((-g/l)\*sin(ThetaT(t-1)))-((c/(m\*l))\*ThetaDT(t-1));

ThetaDT(t)=ThetaDDT(t)\*T\_S+ThetaDT(t-1);

ThetaT(t)=ThetaDT(t)\*T\_s+ThetaT(t-1);

HorzDistMes(t)=(l\*sin(ThetaT(t)))+normrnd(0,0.5);

end

figure;plot(Time,ThetaT,Time,ThetaDT,Time,HorzDistMes);

legend('True theta','True theta dot','measured distance(m)');

xlabel('time(seconds)');title('Model results')

%% Initialize Extended Kalman filter

X = zeros(T,2)'; %Initialize State Vector

X(:,1)=[1.6;0];

P = eye(2); %Initialize covariance matrix

%F=[1,T\_s;(-T\_s\*(g/l)\*cos(X(1,1))),(1-T\_s\*(c/(m\*l)))];

%H=[l\*cos(X(1,1)),0];

Q=[0.0001,0;0,0.0001];

R=0.25;

for k=2:T

%Predict State

X(:,k)=[X(1,k-1)+(T\_s\*X(2,k-1));X(2,k-1)-(T\_s\*(g/l)\*sin(X(1,k-1)))-(T\_s\*(c/(m\*l))\*X(2,k-1))];

%Calculate prediction jacobian and Predict Error covariance

F=[1,T\_s;(-T\_s\*(g/l)\*cos(X(1,k-1))),(1-T\_s\*(c/(m\*l)))];

P=F\*P\*F'+Q;

thetapredict(k)=X(1,k);

thetadotpred(k)=X(2,k);

%Innovation

r=HorzDistMes(k)-(l\*sin(X(1,k)));

%Calculate measurment jacobian and innovation covariance

H=[l\*cos(X(1,k)),0];

S=H\*P\*H'+R;

%Calculate Kalman Gain

K=P\*H'\*inv(S);

%Posterior state estimate

X(:,k)=X(:,k)+(K\*r);

%posterior error covariance

P=(eye(2)-(K\*H))\*P;

ThetaE(k)=X(1,k);

ThetaDE(k)=X(2,k);

end

figure;plot(Time,ThetaT,Time,ThetaE,Time,thetapredict);

legend('true','est','predict');xlabel('Time (seconds)');ylabel('Angle (Radians)');title('Non-Forced EKF Results')

